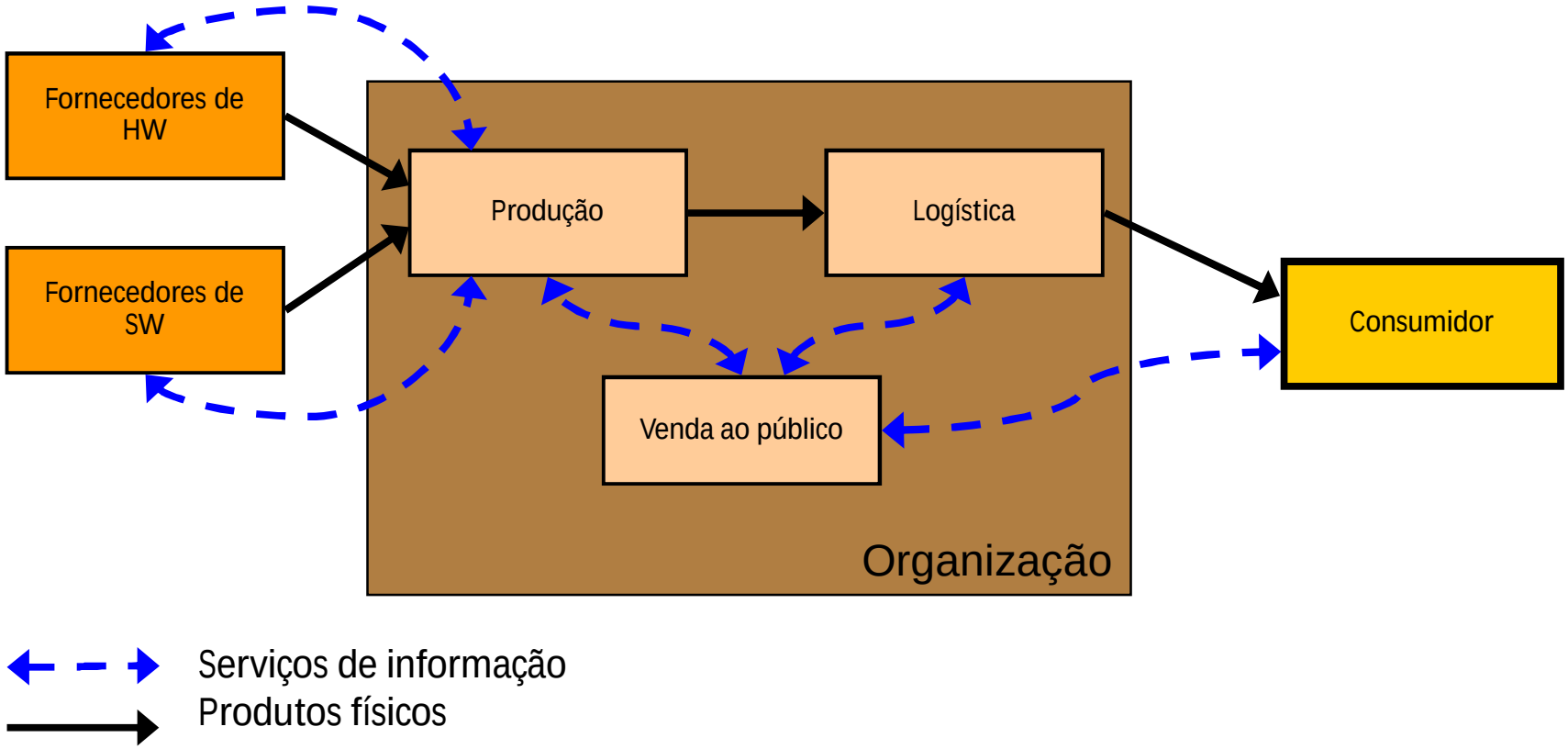
WebServices

Rodrigo Assad

ARQUITETURA ORIENTADA A SERVIÇOS - MOTIVAÇÃO



- Desafios: interoperabilidade de serviços heterogêneos
- Abordagem: serviços voltados para integração
- Resultados esperados: reuso de soluções, agilidade no desenvolvimento

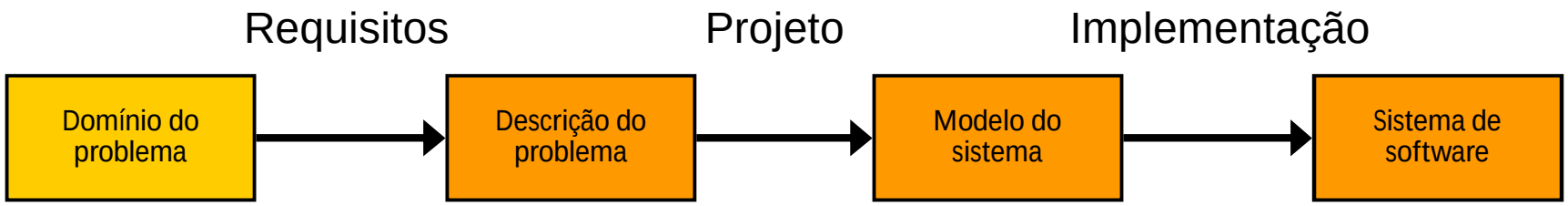
ARQUITETURA ORIENTADA A SERVIÇOS – DEFINIÇÕES

- Arquitetura Orientada à Serviços é um paradigma para organização e utilização de funcionalidades que podem estar sob o controle de diferentes domínios. [OASIS]

- Arquitetura de software é a estrutura (ou estruturas) do sistema, englobando (i) elementos de software, (ii) as propriedades visíveis destes elementos e (iii) as relações entre eles. [BaCl03]

- Software com serviço é um modelo de distribuição de software onde a aplicação é hospedada por um provedor e disponibilizada para consumidores através de uma rede, frequentemente a Internet.

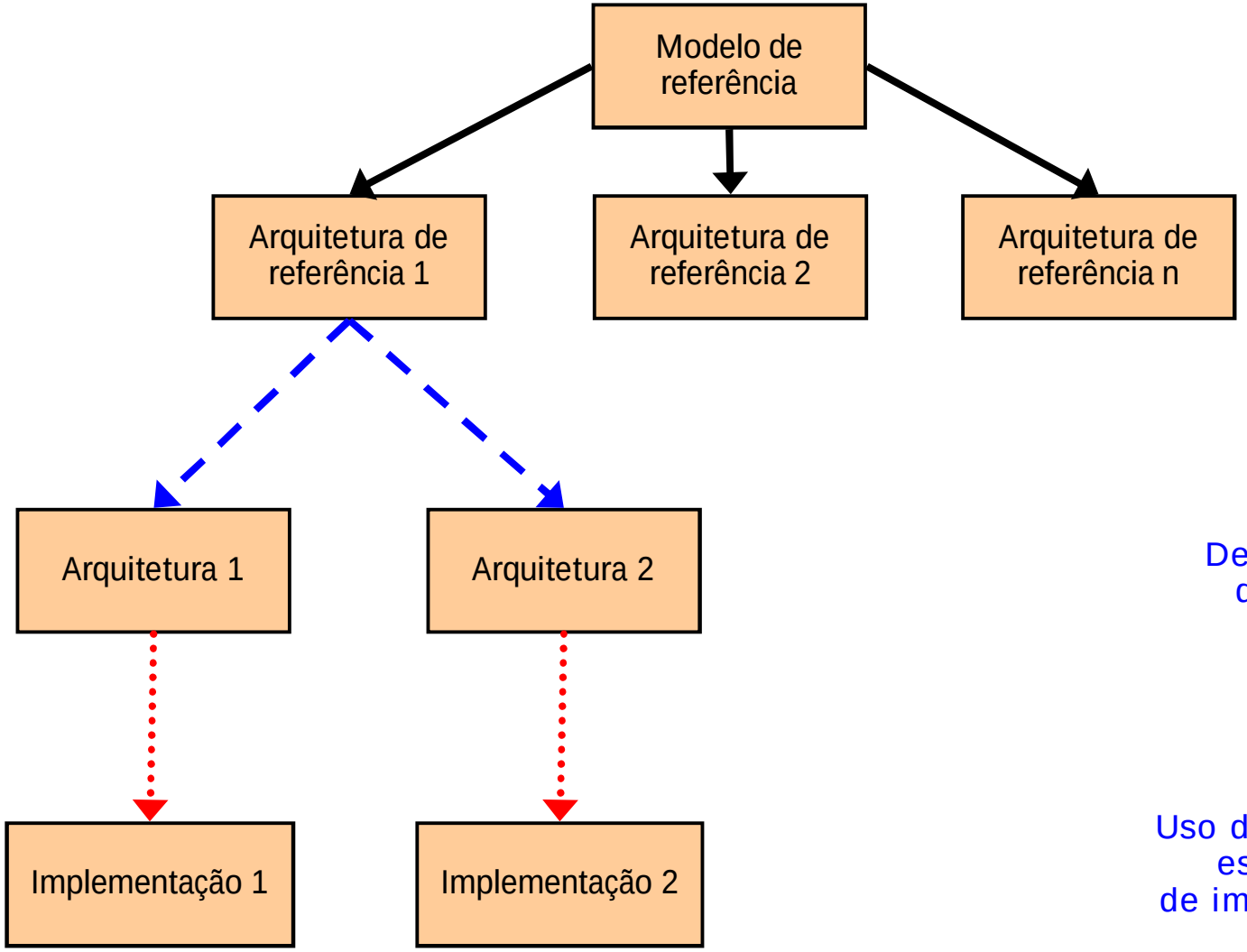
ORIENTAÇÃO À SERVIÇOS - DESENVOLVIMENTO DE SOFTWARE



- Desenvolvimento de software pressupõe o uso de métodos, conceitos, procedimentos e ferramentas
- Métodos de desenvolvimento orientados à serviços usam:
  - Conceitos específicos de modularização para a descrição do problema, em **Requisitos**
  - Conceitos de abstração e estruturação de funcionalidades na forma de serviços, em **Projeto**
  - O uso de procedimentos e ferramentas tecnológicas para a criação do software, em **Implementação**

- Orientação à Objetos e Orientação à Serviços
  - Paradigmas distintos para modelagem e análise de sistemas
  - Focos:
    - OO com foco em mesclar função e dados
    - Serviço como abstração de funções de negócio
  - Acesso à funcionalidades
    - OO, via instância
    - SOA, acesso a serviço já existente
- No entanto...
    - Arquiteturas orientadas à serviço são implementadas com tecnologias orientadas à objeto. Contra-senso?

MODELO DE REFERÊNCIA SOA

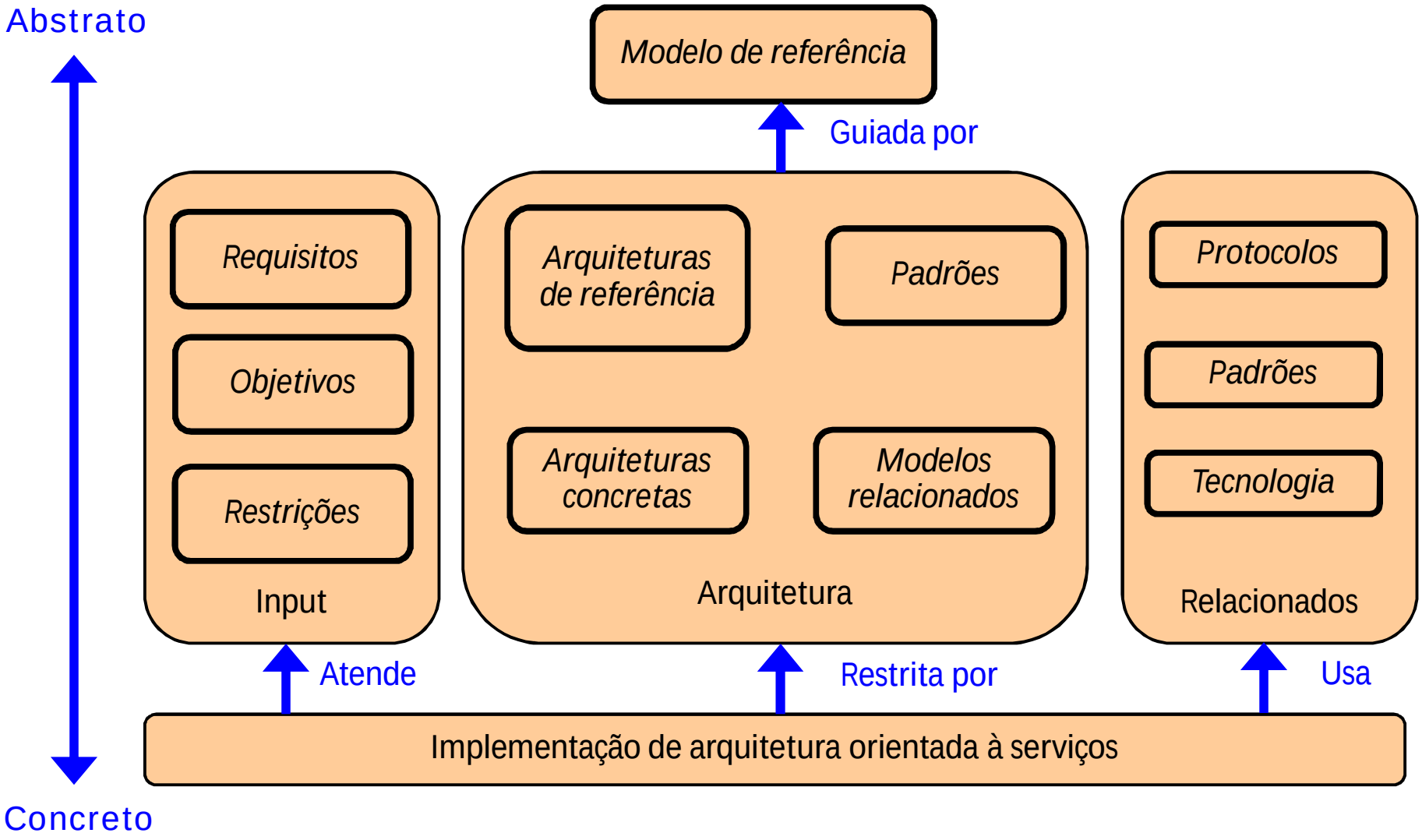


Reuso de alto nível

Detalhamento da solução

Uso de tecnologias específicas de implementação

MODELO DE REFERÊNCIA SOA



- SOA Service Oriented Architecture Não é uma tecnologias específicas, mas pode ser visto como um conjunto de tecnologias, as mais comuns são:

  - REST - Representational State Transfer
  - RPC - Remote Procedure Call
  - DCOM - Distributed Component Object Model
  - CORBA - Common Object Request Broker Architecture
  - Webservices

MODELO DE REFERÊNCIA SOA

- SOA organiza funcionalidades em um ambiente distribuído
  - Casamento entre **necessidade** e **funcionalidade**

- Funcionalidades podem ter diferentes granularidades
  - Específica para uma classe de problemas (necessidades)
  - Pode variar de uma função a uma aplicação
  - Funcionalidades simples podem ser combinadas para atender a necessidades mais complexas
    - Orquestração de serviços (Service Orchestration)
    - Coreografia de serviços (Service Choreography)

- Conceitos de SOA: Visibilidade
  - Capacidade, da necessidade ou serviço, de ser “visível”
  - Feito através de descrições:
    - Funcionalidade
    - Requisitos técnicos
    - Restrições e políticas de uso

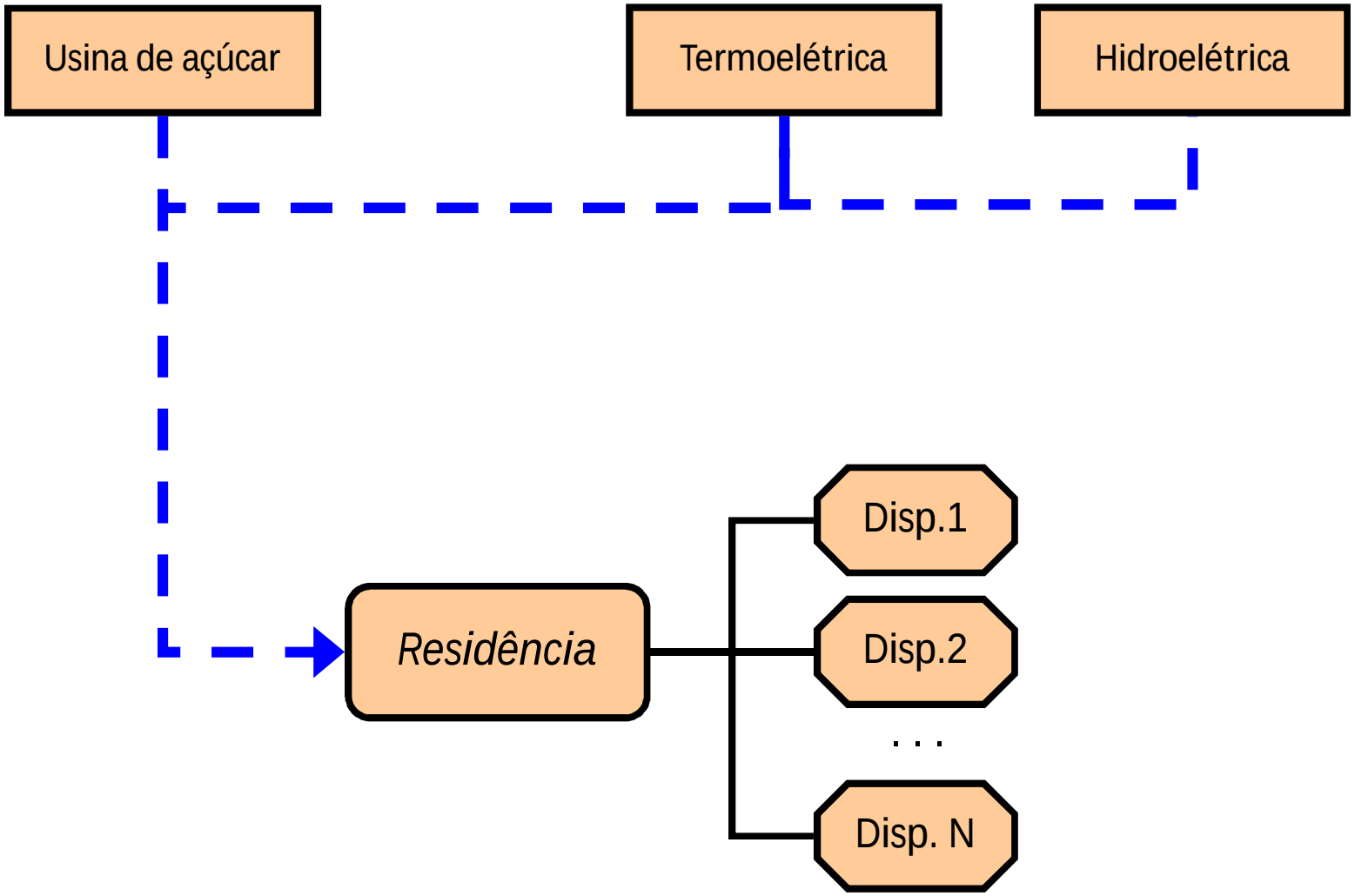
MODELO DE REFERÊNCIA SOA

- Conceitos de SOA: Interação
  - Uso da funcionalidade
  - Acontece dentro de um contexto de execução
    - Padronização de formas de comunicação
    - Ponto de concentração para implementação de políticas e restrições

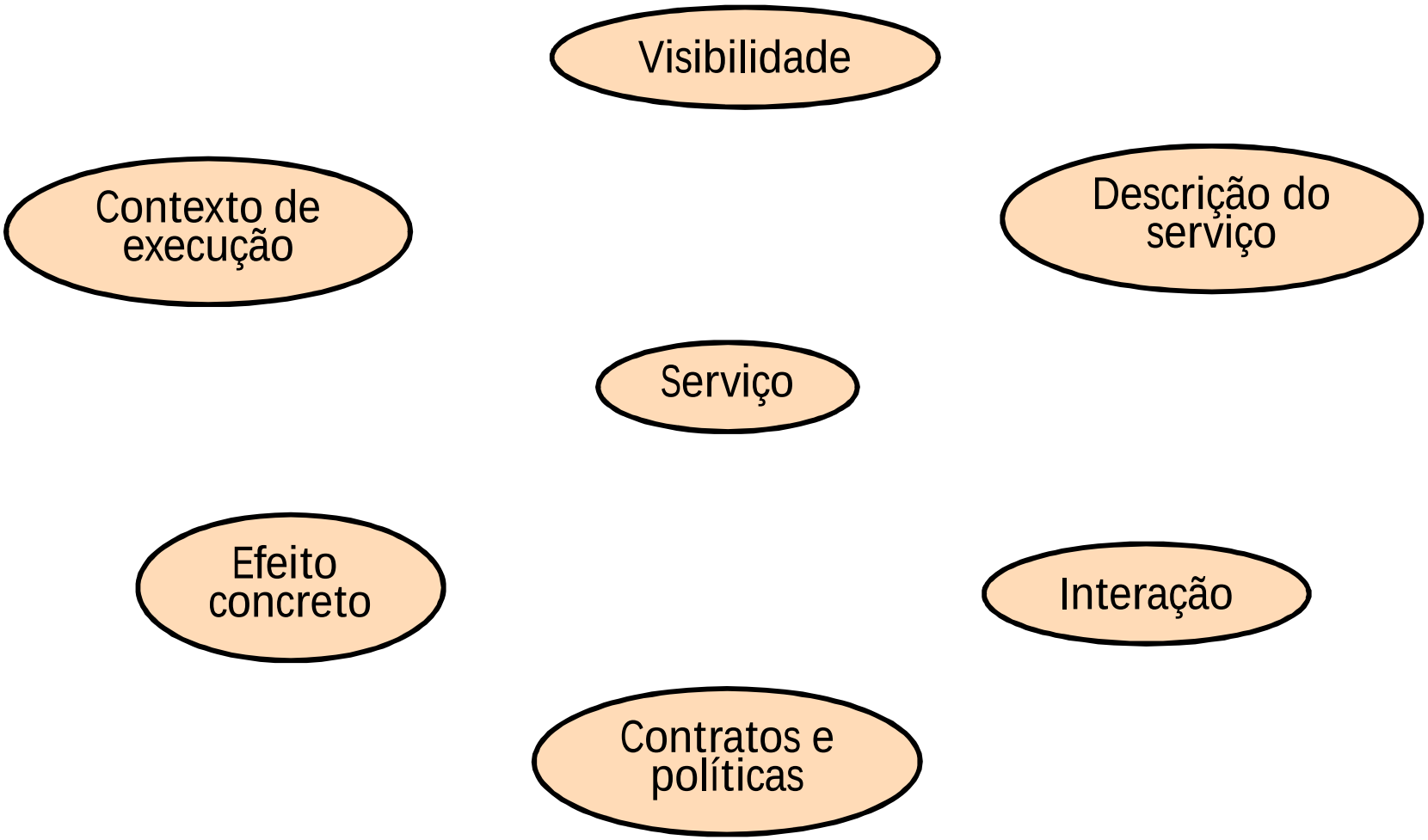
- Conceitos de SOA: Resultados de interação
  - Efeito da funcionalidade sobre um estado compartilhado
  - Perceptível para envolvidos compartilhando mesmo contexto de execução

- Conceitos de SOA: Serviço
  - Capacidade de executar a funcionalidade
  - A especificação da funcionalidade
  - A oferta, a capacidade de prover a funcionalidade

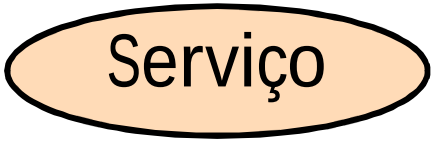
MODELO DE REFERÊNCIA SOA – EXEMPLO DE ARQUITETURA



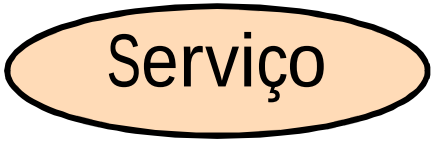
MODELO DE REFERÊNCIA SOA – CONCEITOS



MODELO DE REFERÊNCIA SOA – CONCEITOS

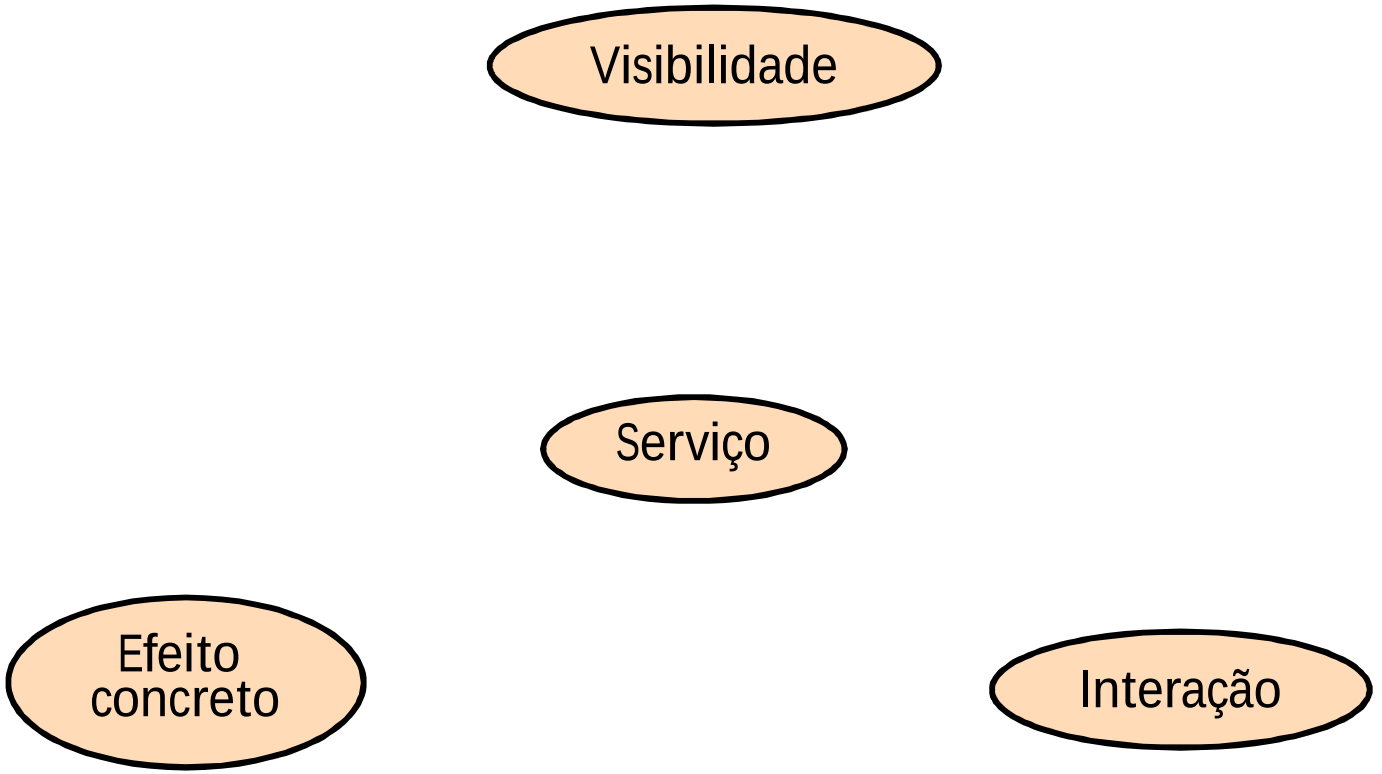


- Mecanismo de acesso à funcionalidade
  - Interface específica
  - Restrições e políticas
- Implementação não é visível para o consumidor, exceto por
  - Modelo de informação
  - Modelo de comportamento
- Efeitos da interação
  - Resposta a requisição de informação
  - Mudança de estado compartilhado
  - Combinação dos acima



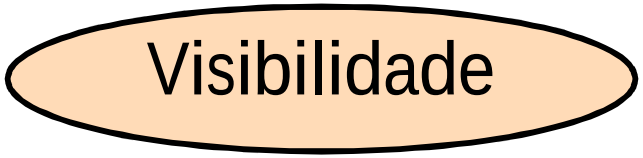
- Exemplo: Serviços de Proteção ao Crédito On-line
  - Mecanismo de acesso à funcionalidade
    - Interface específica on-line
    - Restrições e políticas: autenticação exigida
  - Implementação não é visível para o consumidor, exceto por
    - Modelo de informação: registros com informações
    - Modelo de comportamento: requisição e resposta
  - Efeitos da interação
    - Resposta a requisição de informação: cadastro de indivíduo
    - Mudança de estado compartilhado: log da consulta, visível por consultas subseqüentes

MODELO DE REFERÊNCIA SOA – DINÂMICA DOS SERVIÇOS

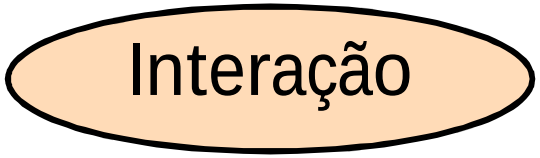


Visibilidade

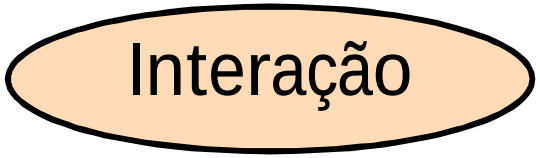
- Ciência
  - Conhecimento sobre a existência de outras partes
  - Informações sobre o serviço em mecanismo de descoberta
- Intenção
  - Interação limitada pela intenção dos participantes
  - Intenção modelada em políticas de serviço
- Acessibilidade
  - Canal de comunicação entre provedor e consumidor
  - Pré-suposto para interação



- Exemplo: DNS (Domain Name System)
  - Ciência
    - Consumidor precisa ter endereço de servidor DNS
    - Hierarquia de servidores (root, domínio, subdomínio)
  - Intenção
    - Tentativa de acesso a URL
    - Política de resposta a requisições, proteção à ataques
  - Acessibilidade
    - Distribuição de informação em servidores
    - Cache distribuído, balanceamento de carga



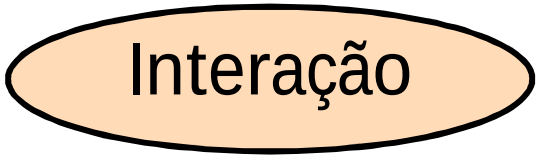
- Modelo de informação
  - Aspectos de estrutura caracterizam dados trocados durante a interação com o serviço em termos de codificação e organização
  - Codificações como ASCII, Unicode, binário
  - Campos de dados em aplicações específicas
  - Modelo semântico, com definições de termos e conceitos usados
  - Implementações podem ser variar em complexidade e abordagem de representação



- Modelo de comportamento : conhecimento sobre as ações de uma interação e suas dependências temporais

- Modelo de ação
  - Caracterização da seqüência de ações, efeitos sobre estados compartilhados e dependências para a execução de ações dentro de contextos diferentes

- Modelo de processo
  - Caracterização das relações e aspectos temporais das ações e eventos associados à interação com um serviço

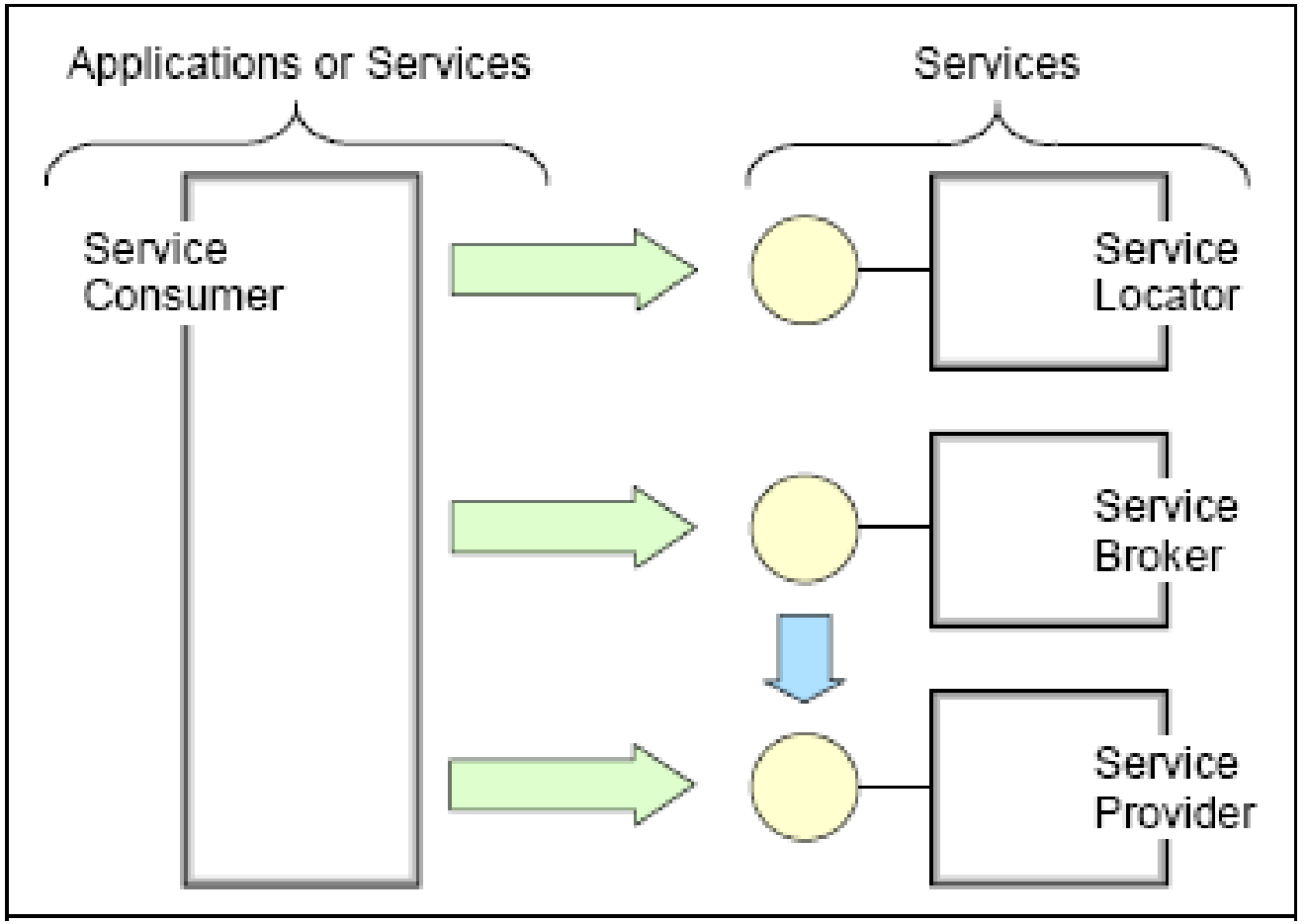


- Exemplo: Busca automatizada de ofertas
  - Modelo de informação
    - Estrutura de dados compartilhadas para cada site de ofertas
    - Uso de campos específicos para identificar conteúdo
    - Semântica compartilhada em ***ontologias***
  - Modelo de comportamento
    - Modelo de ação definindo passos para garantir validade de ofertas e consistência das ações
    - Modelo de processo organizando limites de tempo e coordenação geral entre interações de um único serviço

DESIGN ORIENTADO A SERVIÇO

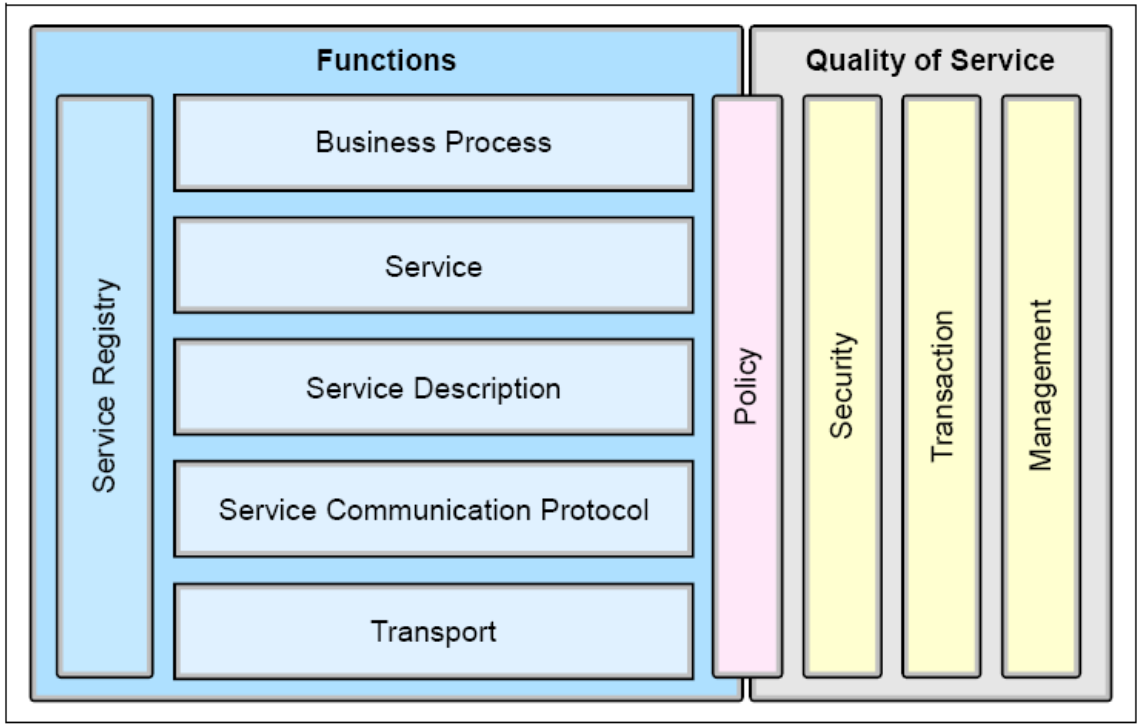
- Componentes:
  - Serviços: entidades lógicas definidas pelas suas interfaces públicas
  - Provedor de serviços (service provider): a entidade de software que implementa o serviços
  - Serviço consumidor (service consumer): entidade de software que utiliza os serviços do provedor de serviços
  - Serviço de localização: por se tratar de um modelo distribuído deve-se ter um sérvio no qual um provedor de serviços se registra e provê suas interfaces e estas podem ser consultadas pelos serviços de consumo.
  - Broker: um tio de serviço especial que permite o encaminhamento de requisições de um serviço a outro.

DESIGN ORIENTADO A SERVIÇO



ARQUITETURA ORIENTADA A SERVIÇO

- Arquiteturas orientadas a serviços são sistemas distribuídos que interoperam, onde cada sistema oferece funcionalidades para os demais.



ARQUITETURA ORIENTADA A SERVIÇO

- Os aspectos funcionais incluem:
  - Transporte: é o mecanismo que permite que os dados das requisições sejam encaminhados do service consumer para o service provider e as respostas do service provider para o service consumer.
  - Service communication protocol: é o protocolo de comunicação acordados entre aspartes
  - Service description: e o esquema acordado entre as partes que descreve o que o serviço é, como ele pode ser invocado e quais dados devem ser passados para invocar o serviço.
  - Service: descreve o serviço disponível.
  - Business proccess: define um conjunto de processos que quando envocados em uma seqüência particular baseando-se em uma seqüência de regras atende as regras de negócios definidas.
  - Service Registry: consiste do repositorio com a descrição dos dados que serão utilizados pelos provedores de serviço (service providers) para publicar seus serviços.

ARQUITETURA ORIENTADA A SERVIÇO

- Aspectos relacionados Qualidade do service:
  - Policy: conjunto de condições ou regras nas quais um provedor de serviço (service provider) disponibiliza seus serviços para os consumidores. Existem aspectos de uma police que são funcionais, aspectos de qualidade de serviço.
  - Security: conjunto de regras que podem ser aplicados para identificação, autorização e controle de acesso para os consumidores.
  - Transaction: conjunto de atributos que é aplicado a um grupo de serviços afim de ter uma resposta consistente. Por exemplo se uma funcionalidade relacionada ao negócio para ser executada é necessárioq eu ela passe por quatro serviços, então ela so pode ser considerada finalizada quando passar pelos quatro elementos,
  - Managment: conjunto de atributos atribuídos ao gerenciador que devem ser providos para o service provider ou service consumer.

Document Type Definition (DTD)

- Seu objetivo é descrever os elementos e atributos que compões o XML
- Uma aplicação pode verificar se todos os campos do XML recebido estão presentes e no formato esperado.
- Pode ser declarado em uma linha dentro do próprio XML referenciada pela TAG *DOCTYPE* ou em um arquivo separado.
- Uma alternativa para o uso dos DTD´s é usar o XML Schema.

Document Type Definition (DTD)

```
<?xml version="1.0"?>
<!DOCTYPE mail [
  <!ELEMENT mail (de,para,assunto,mensagem)>
  <!ELEMENT de       (#PCDATA)>
  <!ELEMENT para     (#PCDATA)>
  <!ELEMENT assunto  (#PCDATA)>
  <!ELEMENT mensagem (#PCDATA)>
]>
<mail>
<de> Bob Silva</de>
<para> Alice Maria</para>
<assunto> aula a noite </assunto>
<mensagem> Alice hoje a noite tera aula?</mensagem>
</mail>
```

Atividade: abrir o arquivo acima em um browser

Document Type Definition (DTD)

- !DOCTYPE mail define que mail é o elemento root para este documento
- !ELEMENT mail define que o elemento mail contém quarto elementos: " de,para,assunto,mensagem ".
- !ELEMENT de define que o elemento de é do tipo "#PCDATA".
- !ELEMENT para define que o elemento para é do tipo "#PCDATA".
- !ELEMENT assunto que o elemento assunto é do tipo "#PCDATA".
- !ELEMENT mensagem define que o elemento mensagem é do tipo "#PCDATA".

Document Type Definition (DTD)

Para separar o DTD do arquivo XML se deve colocar na TAG DOCTYPE o nome do arquivo referenciado
<!DOCTYPE elemento-root SYSTEM "arquivo">.

```
<?xml version="1.0"?>

<!DOCTYPE mail SYSTEM “mail.dtd”>

<mail>

<de> Bob Silva</de>

<para> Alice Maria</para>

<assunto> aula a noite </assunto>

<mensagem> Alice hoje a noite tera aula?</mensagem>

</mail>
```

Arquivo mail.dtd

```
  <!ELEMENT mail (de,para,assunto,mensagem)>

  <!ELEMENT de       (#PCDATA)>

  <!ELEMENT para     (#PCDATA)>

  <!ELEMENT assunto  (#PCDATA)>

  <!ELEMENT mensagem (#PCDATA)>
```

Document Type Definition (DTD)

- Sem valor: deve ser usa a categoria EMPTY

```
<!ELEMENT nome-elemento EMPTY>

Exemplo:

<!ELEMENT br EMPTY>
```

Document Type Definition (DTD)

- Elementos que devem ter seu valor considerado pelo parser: devem conter a TAG PCDATA

```
<!ELEMENT nome-elemento (#PCDATA)>


Exemplo:


<!ELEMENT assunto (#PCDATA)>
```

Document Type Definition (DTD)

- Elementos que podem conter qualquer coisa: se usa a TAG ANY

```
<!ELEMENT nome-elemento ANY>


Exemplo:


<!ELEMENT note ANY>
```

Document Type Definition (DTD)

- Elementos que tenham filhos: se passa os elementos como parâmetros. Os elementos filhos devem ser descritos logo após a declaração da lista com os seus nomes.

```
<!ELEMENT nome-elemento (filho1)>

ou

<!ELEMENT nome-elemento (filho1,filho2,...)>


Exemplo:


<!ELEMENT mail (de,para,assunto,mensagem)>
```

Document Type Definition (DTD)

- Se pode usar expressões regulares para se representar a obrigatoriedade da existência de no mínimo um elemento, um ou mais e etc. A seguir temos vários exemplos com estas declarações.

Deve conter apenas um elemento

```
<!ELEMENT nome-elemento (nome-filho)>

Exemplo:

<!ELEMENT mail (para)>
```

Deve conter no mínimo um elemento

```
<!ELEMENT nome-elemento (nome-filho+)>

Example:

<!ELEMENT mail (para+)>
```

Document Type Definition (DTD)

Deve conter zero ou mais ocorrências do elemento

```
<!ELEMENT nome-elemento (nome-filho*)>

Exemplo:

<!ELEMENT mail (para*)>
```

Deve ocorrer uma ou zero ocorrência do elemento

```
<!ELEMENT nome-elemento (nome-filho?)>

Exemplo:

<!ELEMENT mail (para?)>
```

Deve conter um ou outra ocorencia do elemento

```
Exemplo:

<!ELEMENT mail (de,para,assusnto,(mensagem|texto))>
```

Document Type Definition (DTD)

Pode ocorre de forma mista

```
Exemplo:

<!ELEMENT mail (#PCDATA|de|para|assusnto|mensagem)*>
```

Neste exemplo o campo mail pode ter zero ou mais ocorencias de campo dados ou de elementos `de`,`para`,`assusnto`,`mensagem`.

Document Type Definition (DTD)

- Atributos: são valores extra a informações que os elementos possuem

  A declaração de atributos possui a seguinte sintaxe

```
<!ATTLIST nome-elemento nome-atributo tipo-atributo valor-
default>

Exemplo de DTD:

<!ATTLIST livro type CDATA "papel">

XML resultante:

<livro type="papel" />
```

Document Type Definition (DTD)

Segundo a especificação de XML o campo tipo-atributo pode ser:

Tipo	Descrição
CDATA	A valor é um dado
(en1|en2|..)	O valor pode ser qualquer um da lista
ID	O valor e um identificador único
IDREF	O valor é o ID de outro elemento
IDREFS	O valor é uma lista deoutros ID´s
NMTOKEN	O valor é um nome XML válido
NMTOKENS	O valor é uma lista de nomes XML válidos
ENTITY	O valor é em branco
ENTITIES	O valor emu ma lista de entidades
NOTATION	O valor é o nome de uma notação
xml:	O valor é um valor pre-definido no XML

Document Type Definition (DTD)

Segundo a especificação de XML o campo valor-default pode ser:

Valor	Descrição
value	Valor default do atributo
#REQUIRED	O atributo e requerido
#IMPLIED	O atributo nao é requerido
#FIXED *value*	O atributo possui um valor fixo

Document Type Definition (DTD)

- Entidades: São referencias a textos a caracteres pré-definidos

```
Exemplo de DTD:

<!ENTITY escritor "Jose Maria">

<!ENTITY copyright "CESAR.edu">

XML exemplo:

<author>&escritor;&copyright;</author>
```

```
Exemplo de DTD:

<!ENTITY escritor SYSTEM "http://www.cesar.edu.br/escritor.dtd">

<!ENTITY copyright SYSTEM "http://www.cesar.edu.br/escritor.dtd">

XML exemplo:

<author>&escritor;&copyright;</author>
```

- PCDATA: significa os valores que os dados que passaram pelo parser possuem
- CDATA: valores que não devem ser considerados pelo parser.

XML Schema

- Definir como os elementos devem aparecer no documento
- Definir como os atributos devem aparecer no documento
- Definir quais elementos são elementos filhos
- Definir a ordem dos elementos filhos
- Definir o número de elementos filhos
- Definir quando um elemento é em branco ou nulo e quando ele deve incluir texto
- Definir o tipo de dados de cada elemento
- Definir os valores default e fixos dos elementos e atributos

XML Schemas suporta tipos de dados

- Uma das grandes vantagens do XML Schema é que ele da suporte a tipos de dados. Eles são importantes porque:
  - É mais fácil de interpreter os valores do document original
  - É mais fácil de validar a corretude dos dados recebidos
  - É mais fácil de trabalhar com dados que venham ou que devam ser inseridos no banco de dados
  - É mais fácil definir as restrições aos tipos de dados
  - É mais fácil de definir o formato dos dados
  - É mais fácil de converter tipos de dados

XML Schemas usa a Sintaxe do XML

- Os XML Schemas são escritos em XML e não em uma outra linguagem;
- Alguns benefícios são:
  - Não é necessário aprender outra linguagem
  - Se pode usar o mesmo edito XML para as duas atividades
  - Se pode usar o mesmo parser XML para validar o XSD

XML Schemas tentam implementar corretude na interpretação dos dados

- XSD tenta garantir que os dados enviados por um emissor a um destinatário será interpretados da mesma maneira,isso tenta garantir que os dados sejam interpretados corretamente.
  - Usando XSD o emissor pode descrever os dados da forma que o destinatário vai comprende-lo. Um exemplo é o formato de data, a data 04-09-2004 pode ser interpretada como 04 do setembro de 2004 ou como 09 de abril de 2004. No entanto com a construção:
-
- <date type="date">2004-09-04</date>
-
  - Se garante que o conteúdo do tipo data sera interpretado das mesma forma pelo emissor e destinatário, visto que o tipo data tem o formato "YYYY-MM-DD".

XML Schemas são extensíveis

- Eles são extensíveis porque são escritos em XML, algumas de suas vantagens são:
  - Reuso de Schemas
  - Se pode criar tipos de dados a partir de tipos de dados primitivos
  - Se pode referencias múltiplos schemas em um único documento
  -
  - Por usar as mesmas definições aplicadas ao XML , possuir um único eleento root e etc os XSD são bastante flexíveis e fáceis de serem validados.

Exemplo

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.cesar.edu.br/XMLSchema"
targetNamespace="http://www.cesar.edu.br"
xmlns="http://www.cesar.edu.br"
elementFormDefault="qualified">
<xs:element name="mail">
   <xs:complexType>
    <xs:sequence>
            <xs:element name="de" type="xs:string"/>
            <xs:element name="para" type="xs:string"/>
            <xs:element name="assunto" type="xs:string"/>
            <xs:element name="mensagem" type="xs:string"/>
    </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

Sintaxe

- A TAG schema representa o elemento root.
- A TAG xmlns:xs="http://www.cesar.edu.br/XMLSchema informa de onde os tipos de dados e elementos estão definidos. Eles devem ter o prefixo xs.
- Define que os elementos de, para, assunto e mensagem estão definidos em no namespace targetNamespace="http://www.cesar.edu.br".
- A TAG xmlns="http://www.cesar.edu.br" representa o namespace default.
- elementFormDefault="qualified" define que todos os elementos devem estar descritos no namespace.

Elementos

- Os elementos podem ser simples e complexos.
  - Os elemento simples são baseados nos tipos de dados primitivos suportados, já os complexos definem estruturas de dados mais complexas.
  - Para se definir um elementos simples se usa a construção: <xs:element name="xxx" type="yyy"/>. Os tipos de dados primitivos são.
    - xs:string
    - xs:decimal
    - xs:integer
    - xs:boolean
    - xs:date
    - xs:time

- Os valores de um elementos podem ser fixos ou possuírem um valor default, para estas construções se usa as palavras default e fixed.
- No caso dos valores default quando não se atribui um valor se assume o valor informado no schema. Já no caso de um valor fixo, não se pode atribuir um outro valor.

- <xs:element name="aluno" type="xs:string" default="fulano"/>
- <xs:element name="aluno" type="xs:string" fixed="fulano"/>

Atributos

- A definição de atributos possui a mesma sintaxe: <xs:attribute name="xxx" type="yyy"/> e os mesmos tipos de dados primitivos. Também possuem a facilidade de definir tipos default e fixos.
- Uma diferença para os atributos é que eles podem ser opcionais ou requeridos. O padrão que eles sejam opcionais mas como no exemplo abaixo se pode deixa-lo como requeridos.

- <xs:attribute name="lang" type="xs:string" use="required"/>

Restrições

Restrição	Descrição
Enumeration	Define uma lista de valores aceitos
fractionDigits	Defini o numero máximo de digitos decimais aceitos.
Length	Define o valor exato do número de caracteres.
maxExclusive	Indica o valor máximo que não pode ser superado sem incluir este valor.
maxInclusive	Indica o valor máximo que não pode ser superado incluindo este valor.
maxLength	Especifica o valor máximo do número de caracteres ou lista aceitos.
minExclusive	Indica o valor mínimo que não pode ser superado sem incluir este valor.
minInclusive	Indica o valor mínimo que não pode ser superado incluindo este valor.
minLength	Específica o tamanho mínimo de caracteres ou lista aceitos.
pattern	Define a sequencia exata que será aceita
totalDigits	Define o número exato de dígitos aceitos.
whiteSpace	Define como os caracteres relacionados aos espaços como Tabulações, espaço em branco e etc serão tratados.

Restrições

```
<xs:element name="password">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:minLength value="5"/>
    <xs:maxLength value="8"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

```
<xs:element name="sexo">
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="masculino|feminino"/>
  </xs:restriction>
</xs:simpleType>
</xs:element>
```

Elementos complexos

- Elementos sem valor (vazios)

```
<produto id="1345"/>
```

Se pode representar esta sentença de duas maneiras.

1 Maneira:

```
<xs:element name="produto">
  <xs:complexType>
    <xs:attribute name="id" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

2 Maneira: Se define um tipo para o elemento e se especifica o tipo.

```
<xs:element name="produto" type="prodtype"/>
<xs:complexType name="prodtype">
  <xs:attribute name="id" type="xs:positiveInteger"/>
</xs:complexType>
```

Elementos complexos

- Elementos que apenas contenham outros elementos

```
<empregado>
<nome>John</nome>
<sobrenome>Smith</sobrenome>
</empregado>
```

1 Maneira de representar:

```
<xs:element name=”empregado">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="sobrenome" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

2 Maneira de represnetar:

```
<xs:element name="person" type="persontype"/>
<xs:complexType name="persontype">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Elementos complexos

- Elementos que contenham apenas texto

```
<comida type="sobremesa">pudim</comida>
```

1 Maneira

```
<xs:element name="comida">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="sobremesa" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

2 Maneira

```
<xs:element name="comida">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="comida" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Elementos complexos

- Elementos que contenham textos e outos elementos

```
<carta>
Prezado amigo <nome> Fulano de Tal </nome>
Seu pedido <id>1256</id>
Sera entregue da data de <data>2008-09-02</data>
</descricao>
```

1 Maneira

```
<xs:element name="carta">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="id" type="xs:positiveInteger"/>
      <xs:element name="data" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

2 Maneira

```
<xs:element name="carta" type="tipocarta"/>
<xs:complexType name="tipocarta" mixed="true">
  <xs:sequence>
    <xs:element name="nome" type="xs:string"/>
    <xs:element name="id" type="xs:positiveInteger"/>
    <xs:element name="data" type="xs:date"/>
  </xs:sequence>
</xs:complexType>
```

Elementos complexos

- Se pode usar indicadores para se especificar ordem que os elementos devem ocorrer , número de ocorrências e grupos.
  - All: indica que os elementos filhos podem ocorrer em qualquer ordem, mas devem ocorrer ao menos uma vez.
  - Choice: indica que se deve aparecer um dos elementos filhos.
  - Sequence: indica que os elementos filhos devem aparecer na ordem pré-definida.
  - maxOccurs: indica a quantidade máxima que um elemento pode ocorrer.
  - minOccurs: indica a quantidade mínima que um elemento pode ocorrer.
  - Group: se pode utilizar a TAG group para definir um tipo que pode ser re-aproveitado em um XML.

SOAP - Simple Object Access Protocol

- SOAP é um protocolo de comunicação que permite que aplicações diferentes em plataforma diferentes se comuniquem, preferencialmente sobre protocolos TCP/IP e conseqüentemente utilizando a internet .
- Ele define um formato de mensagem que é definido em utilizando XML.
- SOAP é independente de plataforma e linguagem de programação.
- SOAP foi criado como uma alternativa a CORBA e RPC para realizar chamadas de aplicações remotas
- Utiliza o protocolo HTTP como base.
- Para se fazer a ligação entre o protocolo HTTP e as mensagens SOAP se definiu um content type no protocolo HTTP chamado de application/soap+xml

SOAP - Simple Object Access Protocol

- Toda mensagem SOA tem o seguinte formato:
  - Um elemento chamando Envelope que identifica o documento como sendo uma mensagem SOAP.
  - Um elemento cabeçalho adicional que contém informações do cabeçalho.
  - Em elemento chamando Body que contém as informações sobre o formato das chamadas e respostas.
  - Opcionalmente um elemento chamado Fault que prove informações sobre possíveis erros que podem acontecer durante o processamento das chamadas.

SOAP - Simple Object Access Protocol

- Alguns pontos importantes sobre a sintaxe das mensagens SOAP.
  - Todas as mensagens devem estar em XML
  - Todas as mensagens precisam ter o mesmo Envelope namespace
  - Todas as mensagens precisam usar o mesmo Encoding namespace
  - As mensagens SOAP não podem ter referencias a DTD
  - As mensagens SOAP não podemos conter informações sobre como processar o XML

SOAP - Simple Object Access Protocol

- Estrutura basica de uma mensagem SOAP

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP - Simple Object Access Protocol

- Ao analisar a estrutura básica acima temos dois pontos importantes a serem observados:
  - xmlns:soap: Reprsenta o namespace que o elemento Envelope esta associado
  - soap:encodingStyle: define as estruturas de dados e os tipos de dados utilizados nas mensagens

SOAP - Simple Object Access Protocol - *Elemento HEADER*

- O elemento Header, deve conter informações específicas da aplicação, tais como autenticação, forma de pagamento e etc. Quando ele existir deve ser um elemento filho do elemento Envelope.
- Existem três tipos de atributos que podem ser utilizados no elemento Header:

  - Actor: por se tratar de um protocolo para a interoperabilidade de sistemas, pode-se assumir que uma mensagem pode passar por vários sistemas antes de chegar ao sistema final, sendo assim o atributo actor pode ser utilizado para se identificar o destino final.
  - mustUnderstand: indica se o processamento do HEaderdeve ser mandatário ou opcional. Quando se tem o valor 1 indica que ele é mandatório e em caso do destino não reconhecê-lo deve ser gerado uma falha.
  - encodingStyle: define o formato de codificação da mensagem.

SOAP - Simple Object Access Protocol - *Elemento Body*

- A mensagem abaixo requisita o preço de um notebook em uma loja de compras.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
<soap:Body>
  <m:GetPrice xmlns:m="http://www.cesar.edu.br/preco">
    <m:Item>notebook</m:Item>
  </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

SOAP - Simple Object Access Protocol - *Elemento Body*

A mensagem abaixo e a resposta com o preço.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
<soap:Body>
  <m:GetPriceResponse xmlns:m="
http://www.cesar.edu.br/preco">
    <m:Price>3456.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

O elemento Fault deve aparecer como um elemento filho do elemento body e somente pode aparecer uma vez na mensagem SOAP.ele possui a seguinte estrutura.

Sub elemento	Descrição
<faultcode>	O código que identifica a falha
<faultstring>	Uma explicação em texto plano sobre a falha
<faultactor>	Informação do que pode ter causaod a falha
<detail>	

Os códigos de erros podem ser:

Erro	Descrição
VersionMismatch	Foi encontrado um namespace invalid para o envelope SOAP
MustUnderstand	Um elemento filho do elemento Header foi encontrado e não foi reconhecido bem como o atributo mustUnderstand estava setado para um
Client	A mensagem foi formada errada ou contém uma informação incorreta
Server	Houve um erro no servidor e ele não conseguiu processar a mensagem

SOAP - Simple Object Access Protocol - *Exemplo*

Requisição

```
POST /NoEstoque HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: TAMANHO MSG

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
 <soap:Body xmlns:m="http://www.cesar.edu.br/estoque">
  <m:PrecoEstoque>
   <m:Produto>notebook</m:Produto>
  </m: PrecoEstoque>
 </soap:Body>
</soap:Envelope>
```

SOAP - Simple Object Access Protocol – *Exemplo*

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: TAMANHO MSG
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
  <soap:Body xmlns:m=" http://www.cesar.edu.br/estoque">
    <m:RespostaPrecoEstoque>
      <m:Preco>2345.90</m:Preco>
    </m: RespostaPrecoEstoque>
  </soap:Body>
</soap:Envelope>
```

WSDL - Web Services Description Language

– WSDL é um padrão para a descrição de webservices, como descrevê-los e chamá-los. Como SOAP sua descrição é feita em XML e também é um padrão definido pelo W3C.

– Um documento WSDL tem a seguinte estrutura:

-

Elemento	Definição
<portType>	A operação provida pelo webservices
<message>	As mensagens utilzadas pelo webservices
<types>	Tipos de dados utilizados pelos webservices
<binding>	Protocolos de comunicação utilizados pelos webservices

WSDL - Web Services Description Language

```
<definitions>
<types>
  Definicao de tipos.....
</types>

<message>
  Definicao de mensagens ....
</message>

<portType>
  Definicao de portas .....
</portType>

<binding>
  Definicao de dos protocolos de comunicação  ....
</binding>

</definitions>
```

WSDL - Web Services Description Language

- O elemento portType é o mais importante pois ele descreve o WebService , as operações providas e as mensagens trocadas. Algumas pessoas comparam ele ao conceito de bibliotecas no processo de desenvolvimento tradicional.
- O elemento message define os tipos de dados.
- O elemento types define os tipos de dados utilizados
- O elemento bindings define o tipo de protocolo utilizado e o formato da mensagem. O elemento binding possui dois atributos, o primeiro deles aponta define qual elemento portType esta associado a este binding e o segundo um nome que pode ser definido pelo usuário e não tem associação alguma com outros elementos.
  - O elemento soap:binding possui dois atributos o estilo e o protocolo de transporte utilizado.
    - O estilo possui dois valores “rpc” ou “document”.
  - O element operation define cada operação especificada pelo element port.

WSDL - Web Services Description

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
 <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
 </operation>
</portType>
<binding type="glossaryTerms" name="b1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
 <operation>
  <soap:operation
   soapAction="http://www.cesar.edu.br/getTerm"/>
  <input>
   <soap:body use="literal"/>
  </input>
  <output>
   <soap:body use="literal"/>
  </output>
 </operation>
</binding>
```